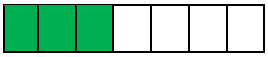
Log4PHP

Version 1.1.0

Niveau requis : 3/7



Mise en place de Log4PHP

Sommaire

I. Préambule

I.I. *Objet*

L'objet de se document est de présenter la mise en place de log dans son application web PHP.
Nous utiliserons Log4PHP.

I.II. *Pré-requis*

Avoir réalisé la formation jusqu'au document [R1] « Mise en oeuvre de Doctrine ».

Par la suite, nous avons renommé le répertoire « web » en « public » pour correspondre à se qui se fait le plus dans les frameworks des applications web tels Zend ou Symfony.

I.III. *Versions du document*

Version	Date	Auteur	Description
1.0.0	01/04/2018	Péquignat.eu	Version initiale du document
1.1.0	13/03/2022	Péquignat.eu	Retrait des références à l'autoentreprise

I.IV. *Documents de référence*

#	Document	Version	Auteur(s)
[R1]	Mise en œuvre de Doctrine	1.2.0	Péquignat.eu

II. Qu'est ce qu'un Logger ?

II.I. *Description*

Un logger est une fonctionnalité permettant d'écrire des messages applicatifs suivant un niveau de criticité dans un ou des flux de sortie.
Habituellement, le logger est paramétré pour écrire dans un fichier sur disque.

Il peut être intéressant aussi d'envoyer un email en cas d'erreur Fatal sur son application Web ou d'enregistrer les logs en base de données.

II.II. *Log4php*

Nous allons mettre en place le log4php d'apache : https://logging.apache.org/log4php/

La dernière version actuelle est la 2.3.0.

III. Objectif

Nous allons mettre en place un logger qui écrit dans un fichier disque, un par jour et avec un envoie de mail en cas d'erreur Fatal.

IV. Installation

Pour l'installation, nous allons utiliser Composer inclut dans Zend Studio 13.6:
Dans le Projet « Cours », double-cliquer sur le fichier « composer.json »

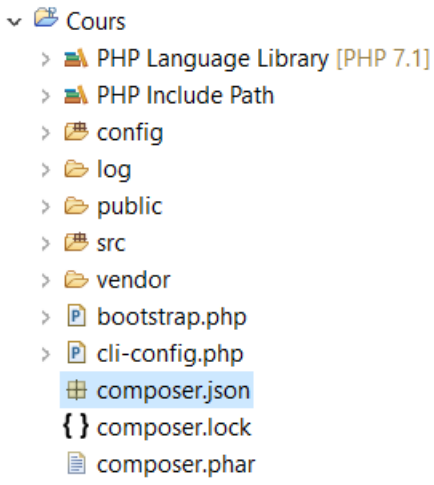


Figure 1 - Fichier composer.json

La page suivante s'ouvre :

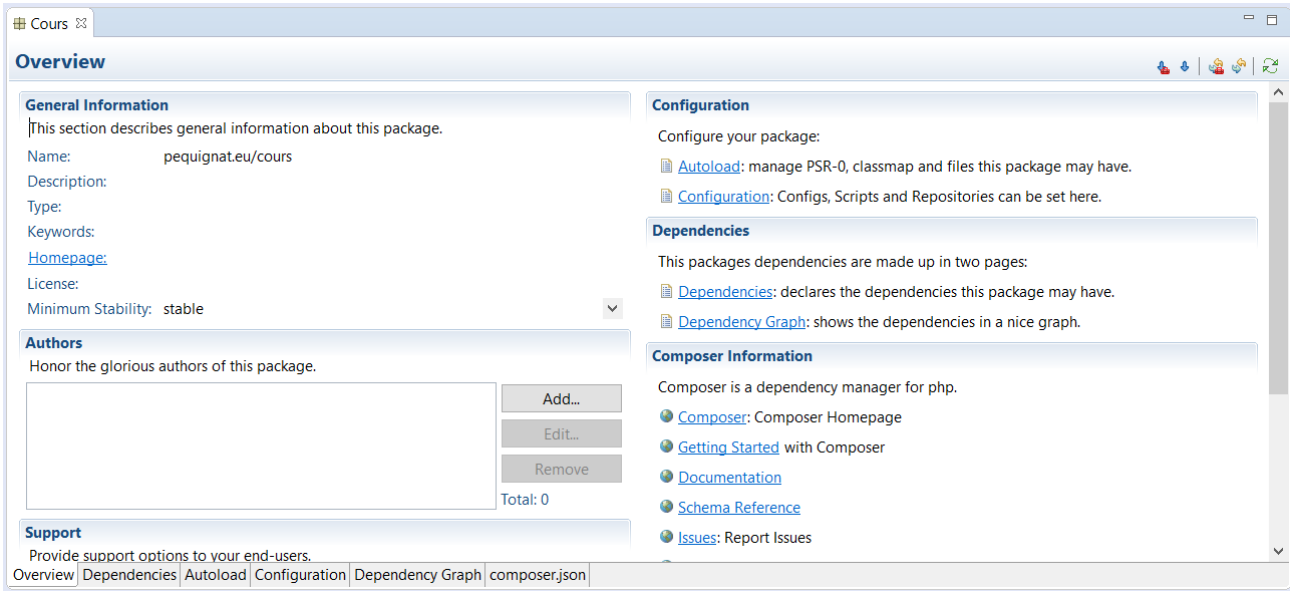


Figure 2 - Overview

Cliquer sur « Dependencies »

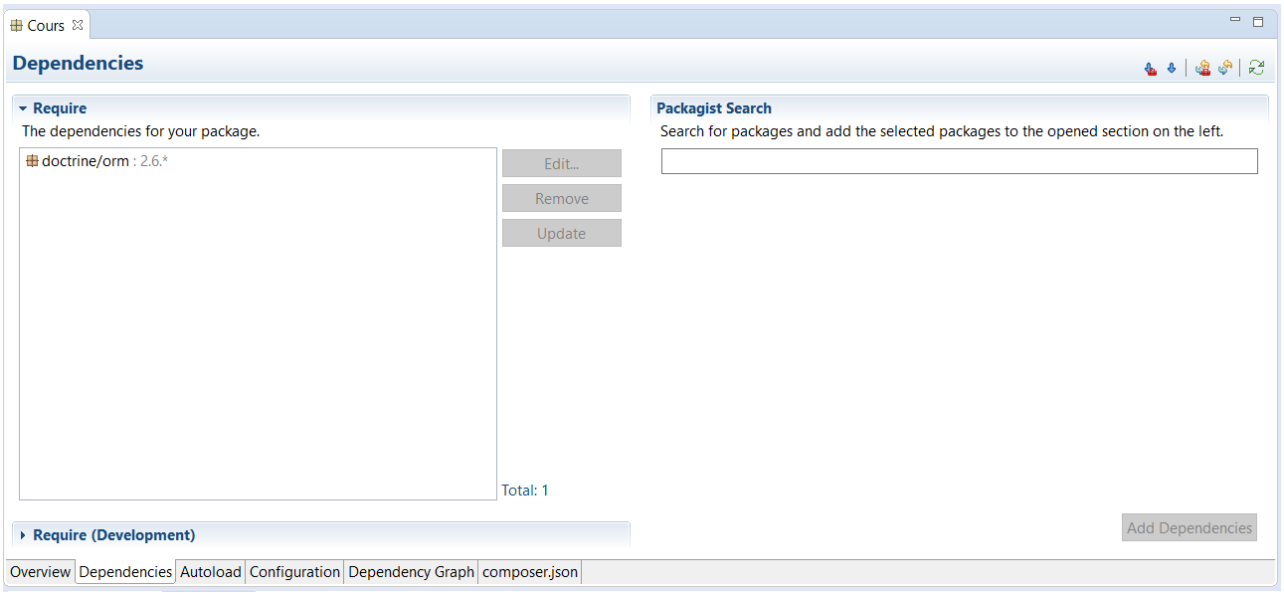


Figure 3 - Dependencies composer.json

Dans la partie de droite « Packagist Search » mettez : « apache/log4php »

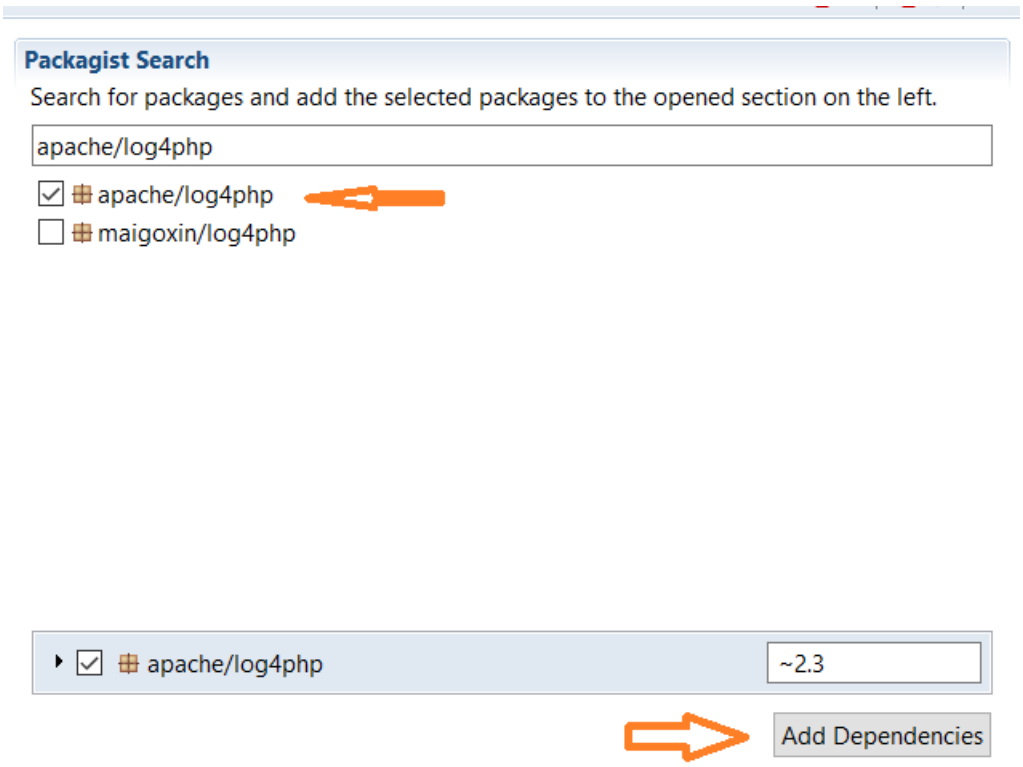


Figure 4 - Ajout de la dépendance apache/log4php

Cliquez sur « Add Dependencies »
Puis CTRL+S (Sauvegarder)
Cliquer sur « Update Dependencies »

Figure 5 - Update Dependencies

V. Préparation

V.I. *Création des répertoires*

Créer un répertoire contenant les futurs logs : « log » à la racine du projet « Cours »

Créer un Sous répertoire « Config » dans le répertoire « Src » qui va contenir la classe de configuration du logger.

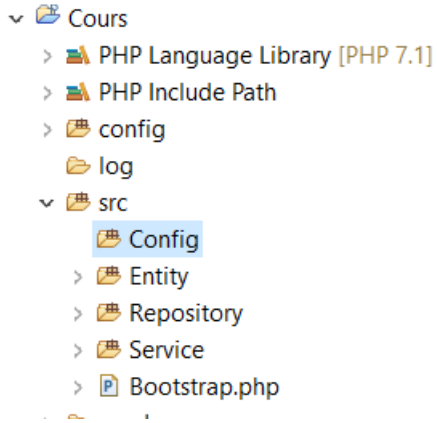


Figure 6 - Création répertoires pour le log

V.II. *Initialisation du PHP.ini*

Vous êtes sous Windows, vous aurez besoin de configurer votre fichier PHP.ini de votre application afin d'envoyer des emails par le logger.

Figure 7 - configuration initiale des mails pour PHP

Configurer suivant votre serveur ou machine de DEV

V.III. *Création de la classe de configuration MailConfig*

Dans le répertoire Config, à la racine nous créons la classe MailConfig.
Cette classe va servir à enregistrer la configuration des mails pour le logger.

```php
<?php


class MailConfig
{
    /**
     * Renvoie les paramètres de mails
```

```
     * @return array
     */
    public static function getMailParams() : array
    {
        return array(
            'from' => 'postmaster@mondomaine.eu', // le FROM : adresse d'origine
            'toLog' => 'support@mondomaine.eu,postmaster@mondomaine.eu' //les TO
: adresses de réceptions des mails de log
        );
    }
}
```

V.IV. *Création de la classe MyLoggerConfigurator*

Dans le répertoire « Src/Config » créer la classe « MyLoggerConfigurator » qui implémente « LoggerConfigurator »

```
<?php
namespace Config;

use \LoggerConfigurator;
use \LoggerHierarchy;
use \LoggerAppenderDailyFile;
use \LoggerAppenderMailEvent;
use \LoggerLayoutPattern;

/**
 * La classe de configuration du logger log4php
 * @author Christophe PEQUIGNAT
 *
 */
class MyLoggerConfigurator implements LoggerConfigurator
{
```

```
    private $fileParams;
    private $mailParams;

    /**
     *
     * @param array $fileParams
     * @param array $mailParams
     */
    public function __construct(array $fileParams, array $mailParams){
        $this->fileParams = $fileParams;
        $this->mailParams = $mailParams;
    }


    /**
     * {@inheritDoc}
     * @see LoggerConfigurator::configure()
     */
    public function configure(LoggerHierarchy $hierarchy, $input = null)
    {
        // Use a different layout for the next appender
        $layout = new LoggerLayoutPattern();
        $layout->setConversionPattern("%d{ISO8601} %level [%logger]
%msg%newline");
        $layout->activateOptions();

        // Create an appender which logs to file
        $appFile = new LoggerAppenderDailyFile('file');
        $appFile->setFile( $this->fileParams['file'] );
        $appFile->setAppend(true);
        $appFile->setThreshold($this->fileParams['level']);
        $appFile->setLayout($layout);
        $appFile->activateOptions();

        // Create an appender which logs to file
```

```
        $appMailEvent = new LoggerAppenderMailEvent('mailEvent');
        $appMailEvent->setLayout($layout);
        $appMailEvent->setThreshold($this->mailParams['level']);
        $appMailEvent->setFrom($this->mailParams['from']);
        $appMailEvent->setTo($this->mailParams['to']);
        $appMailEvent->activateOptions();

        // Add both appenders to the root logger
        $root = $hierarchy->getRootLogger();
        $root->addAppender($appFile);
        $root->addAppender($appMailEvent);
    }

}
```

Cette classe permet d'initialiser les « Appender » Fichier et Mail.

Ici, nous avons choisi de faire un fichier par jour et d'utiliser l'envoie de mail par événement.

V.V. *Mise à niveau du Bootstrap*

Maintenant que nous avons les classes de paramétrage, il convient d'initialiser « configurer » le logger.

Pour cela nous allons ajouter une méthode dans la classe Bootstrap.php à la racine de « Src »

```
<?php
// src/Bootstrap.php
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;
use Config\MyLoggerConfigurator;

class Bootstrap{

    private static $devLogLevel='debug';

    /**
```

```
     *
     * @var bool
     */
    private static $isDevMode = false;


    /**
     *
     * @var EntityManager
     */
    private static $entityManager = null;


    private static function initDevMode() : void {
        self::$isDevMode = in_array( APPLICATION_ENV , array('development'));
    }


    private static function initLogging(){
        // User defined configuration (optional)
        $configuration = array(
            'file' => 1,
            'mailEvent' => 2
        );
        $fileParams=array('file'=>__DIR__.'/../log/' .(self::$isDevMode ? 'dev'
: 'prod') .'/cours.log' , 'level' => self::$isDevMode ? self::$devLogLevel :
'warn');
        $mailsParamsConfig = \MailConfig::getMailParams();
        $mailParams=array('to' => $mailsParamsConfig['toLog'], 'from' =>
$mailsParamsConfig['from'] , 'level' => 'fatal');
        // Passing the configurator as an instance
        Logger::configure($configuration, new MyLoggerConfigurator($fileParams,
$mailParams));
    }


    private static function initDatabase() : void{
        // Create a simple "default" Doctrine ORM configuration for Annotations
        $config = Setup::createAnnotationMetadataConfiguration(array(__DIR__),
self::$isDevMode);
        // or if you prefer XML
```

```
        //$config =
Setup::createXMLMetadataConfiguration(array(__DIR__."/config"), $isDevMode);
        // database configuration parameters
        $conn =
\Doctrine\DBAL\DriverManager::getConnection(Database::getConnectionParams(),
$config);


        // obtaining the entity manager
        self::$entityManager = EntityManager::create($conn, $config);
    }



    public static function init() : void{
        self::initDevMode();
        self::initLogging();
        self::initDatabase();
    }


    /**
     * Donne access à l'EntityManager d'accès à la base
     * @return EntityManager
     */
    public static function getEntityManager() : EntityManager{
        return self::$entityManager;
    }
}
```

Voila, c'est configurer !

VI. Utilisation du logger

VI.I. *Usage*

Maintenant que le logger est configuré, l'usage est assez simple :

```
$logger = Logger::getLogger('ORIGINE');
$logger->info('Bonjour');
```

VI.II. *Exemple*

Voici le fichier index.php présent dans le répertoire « public »

```php
<?php


defined ("APPLICATION_ENV") || define("APPLICATION_ENV",
getenv('APPLICATION_ENV') ?? 'production');
echo 'Environnement : '.APPLICATION_ENV . "<br/>\r\n";


echo 'Initialisation de la connexion à la base de données'. "<br/>\r\n";
require_once('..\bootstrap.php');
use \Service\CourseService;


echo 'Initialisation du logger'. "<br/>\r\n";
$logger = Logger::getLogger('index.php');


$logger->info('Bonjour');


$courseService = new CourseService(Bootstrap::getEntityManager());


echo 'Initialisation des datas'. "<br/>\r\n";
$courseService->initExample();
echo 'Affichage des datas'. "<br/>\r\n";


echo '==>Affichage des Personnes :'."<br/>\r\n";
foreach($courseService->getAllPersons() as $person){
    echo $person."<br/>\r\n";
}
echo "<br/>\r\n";
echo '==>Affichage des Cours :'."<br/>\r\n";
foreach($courseService->getAllCourses() as $cours){
    echo $cours."<br/>\r\n";
}
echo "<br/>\r\n";
echo 'Suppression des datas'. "<br/>\r\n";
$courseService->dropExample();
```

```
$logger->fatal('Contenu du mail Fatal');
```

Cela donne le résultat en développement :

Le fichier de log est généré :

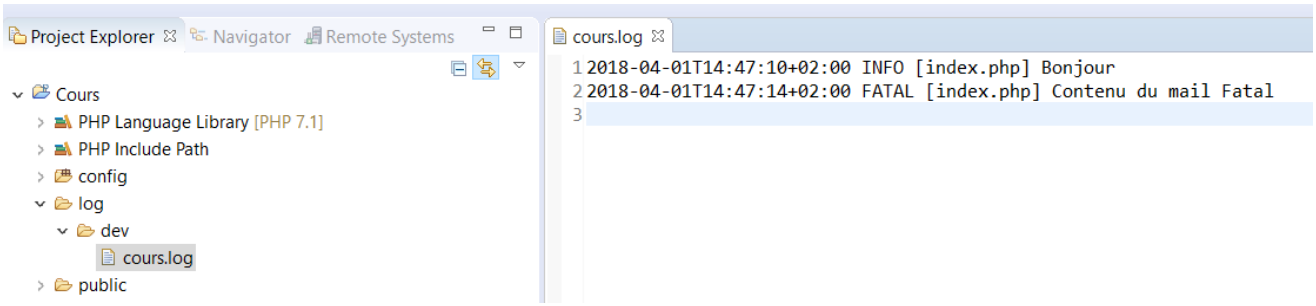


Figure 8 - Fichier log dans dev

Un mail est envoyé :

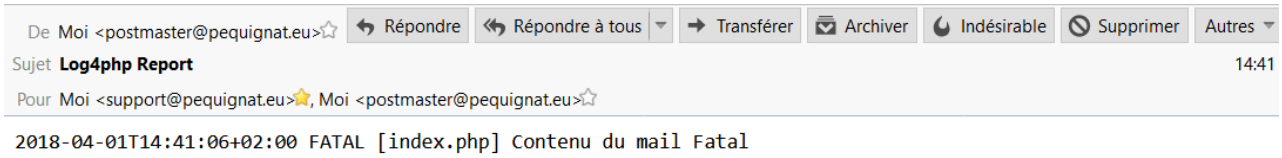


Figure 9 - Report mail de log4php

VII. Fin du document